Self-organising processes of task allocation

Zoethout, K.

Appendix I

Relevant Computer Code

Allocation

```
Procedure TSystem.Allocate(rounde:integer);

var s,count:integer;

begin
  count:=0;
  InitialChoice;
  repeat
  for s:=0 to Agent[0].NrOfSkills-1 do
  begin
    if Agent[0].Skill[s].Active=True AND
       Agent[0].Skill[s].Available=True then
    begin
      ModifyInitialChoice;
      if (NrOfAgentsWithSameHighestSkill(s) > 0) then
      begin
        while ParForm.Task.AvailableActions(s)< NrOfAgentsWithSameHighestSkill(s) do
        begin
          RemLowestAgent(s,LowestAgent(s));
          if ParForm.InflCheckBox.Checked=True then
          begin
           count:=count+1;
           ParForm.CountEdit.Text:=IntToStr(count);
           ShowMessage('Influence');
          end;
        end;
      ActionToAgent(s,rounde);
      end;
    end;
  end;
until(AvailableAgents=0)or(ParForm.Task.NrOfTaskUnitsAllocated=ParForm.Task.NrOfTaskUnit
s);
end;

procedure TSystem.ActionToAgent(s,rounde:integer);

var a,c:integer;
RoundAgent:string;
begin
  for a:=0 to NrOfAgents-1 do
  begin
    if (Agent[a].Skill[s].Choice=True) AND
       (Agent[a].Free=True) then
```

```
    begin
      c:=ParForm.Task.FirstAvailableCycle(s);
      if c< ParForm.Task.NrOfCycles then
      begin
        Agent[a].Free:=False;
        ParForm.Task.Taskunit[c,s].Open:=False;
      ParForm.Task.NrOfTaskUnitsAllocated:=
      ParForm.Task.NrOfTaskUnitsAllocated+1;
      if ParForm.AllCheckBox.Checked=True then
        begin
          RoundAgent:=IntToStr(rounde+1)+ ' / '+IntToStr(a+1);
          ParForm.TTaskFrame.TaskGrid1.Cells[c+1,s+1-tvar]:=RoundAgent;
        end;
      end;
      if ParForm.Task.AvailableActions(s)= 0 then Agent[0].Skill[s].Available:=False;
    end;
  end;
end;
```

Main Process

```
procedure TSystem.Start;

var k,NrOfTasks,rounde:integer;

begin
  remcounter:=0;
  NrOfTasks:=StrToInt(ParForm.NrOfTasksEdit.Text);
  tvar:=-ParForm.Task.NrOfVariation; ///handig bij loop waar tvar:=tvar+
  Task.NrofVariation
  IntTimeTotToZero;
  TotalTimeToZero;
  for k:=0 to NrOfTasks-1 do
  begin
    WTimeTotToZero;
    tvar:=tvar+ParForm.Task.NrOfVariation;
    if tvar>(StrToInt(ParForm.NrofSkillsEdit.Text))then tvar:=tvar-
    StrToInt(ParForm.NrofSkillsEdit.Text);
    NextTask(k);
    rounde:=0;
    while (ParForm.Task.NrOfTaskUnitsDone< ParForm.Task.NrOfTaskUnits) do
    begin
      InteractionTimeToZero;
      WorkTimeToZero;
      if ParForm.CycleCheckBox.Checked=TRUE then
      begin
        ShowMessage ('New Round');
        ParForm.Edit2.Text:=IntToStr(rounde+1);
        ParForm.ShowStatesAgents;
      end;
      if ParForm.AgVarCheckBox.Checked=True then
      Turnoverfit(rounde);//TurnoverNewComerLow(rounde);
      if ParForm.AllocationRadioGroup.ItemIndex =0 then
      Allocate(rounde+k*ParForm.Task.NrOfRounds);
      if ParForm.AllocationRadioGroup.ItemIndex =2 then
      Generalise(rounde+k*ParForm.Task.NrOfRounds);
      if ParForm.AllocationRadioGroup.ItemIndex =1 then Specialise(rounde,k);
```

```
    Perf(rounde+k*ParForm.Task.NrOfRounds,k);
    rounde:=rounde+1;
    SysIntTimeTot:=SysIntTimeTot + SysIntTime;
    WorkTimeTot:=WorkTimeTot+ WorkTime;
  end;
  ParForm.WorkTimeGraph[k,NrOfAgents]:=WorkTimeTot;
  TotalTime:=TotalTime+WorkTimeTot;
  ParForm.WorkTimeGraph[StrToInt((ParForm.NrOfTasksEdit.Text))+
  1,ParForm.System.NrOfAgents]:=TotalTime;
  ParForm.WorkTimeGraph[k+1,ParForm.System.NrOfAgents+1]:=TotalTime;
ParForm.ShowStatesAgents;
 end;
end;
```

Performance

procedure TSystem.Perf(rounde,k:integer);

```
var a: integer;

begin
 for a:=0 to NrOfAgents-1 do
 begin
   if Agent[a].Present=True then
   Agent[a].Perf(rounde,k,a);
   ParForm.PerfGraph[rounde+1,a]:=Agent[a].WorkTime; //graphic van agent
   ParForm.WorkTimeGraph[k,a]   :=Agent[a].WorkTimeTot;
   ParForm.WorkTimeGraph[StrToInt(ParForm.NrOfTasksEdit.Text)+1,a]:=
   Agent[a].TotalTime;
    if Agent[a].WorkTime>WorkTime then WorkTime:=Agent[a].WorkTime;
 end;
 ParForm.PerfGraph[rounde+1,NrOfAgents]:=WorkTime; //graphic van system
end;
```

procedure TAgent.Perf(rounde,k,a:integer);

```
var s: integer;

begin
 for s:=0 to NrOfSkills-1 do
 begin
   if (Skill[s].Choice = TRUE) AND (Skill[s].Active = TRUE) then
   ChoiceTrueActiveTrue(rounde,k,a,s);
   if (Skill[s].Choice = FALSE)AND (Skill[s].Active = TRUE)then
  ChoiceFalseActiveTrue(rounde,a,s);
   if Skill[s].Active = FALSE then ActiveFalse(rounde,s);
 end;
 WorkTimeTot:=WorkTimeTot+WorkTime;
 TotalTime:=TotalTime+WorkTimeTot;
end;
```

procedure TAgent.ChoiceTrueActiveTrue(rounde,k,a,s:integer);

```
var c:integer;
 RelMot,RelExp:real;
 RoundAgent:string;

 begin
   RelMot:=(1-Skill[s].ExpMotRate)*(Skill[s].Motivation/Skill[s].MotMax);
```

```
    RelExp:=Skill[s].ExpMotRate*(Skill[s].Expertise/ Skill[s].ExpMax);
    c:=ParForm.Task.FirstCycleToDo(s);
    if c<ParForm.Task.NrOfCycles then
    begin
      ParForm.Task.TaskUnit[c,s].Done:=True;
      RoundAgent:=IntToStr(rounde+1)+ '/'+IntToStr(a+1);
      ParForm.TTaskFrame.TaskGrid1.Cells[c+1,s+1-tvar]:=RoundAgent;
     ChartForm.AllocationGrid.Cells[c+1+k*ParForm.Task.NrOfCycles,s+1]:=RoundAgent;
     ParForm.Task.NrOfTaskUnitsDone:= ParForm.Task.NrOfTaskUnitsDone+1;
     if ParForm.PerfCheckBox.Checked=True then ShowMessage('Performed');
     Skill[s].Used[rounde]:=TRUE;  ////
     Skill[s].WorkTime:= StrToInt(ParForm.DurEdit.Text)/(RelMot+RelExp);
     WorkTime:=WorkTime+Skill[s].WorkTime; //worktime agent=sum(worktime skills)
     Skill[s].Learning;
     Skill[s].BoredomRecovery;
     ParForm.ExpGraph[rounde+1,a,s]:=Skill[s].Expertise;
     ParForm.MotGraph[rounde+1,a,s]:=Skill[s].Motivation;
     Free:=True;
  end;
end;

procedure TAgent.ChoiceFalseActiveTrue(rounde,a,s:integer);

begin
  Skill[s].Used[rounde]:=False;////
  Skill[s].WorkTime:=0;
  Skill[s].Forgetting;
  Skill[s].BoredomRecovery;
  ParForm.ExpGraph[rounde+1,a,s]:=Skill[s].Expertise;
  ParForm.MotGraph[rounde+1,a,s]:=Skill[s].motivation;
end;

procedure TAgent.ActiveFalse(rounde,s:integer);

begin
  Skill[s].Used[rounde]:=False;////
  Skill[s].WorkTime:=0;
  Skill[s].Forgetting;
  Skill[s].BoredomRecovery;
end;
```

Influence

```
procedure TSystem.SelfOrganisation(rounde,k:integer);

begin
  InitialChoice;
  AllocateSelf(rounde+k*ParForm.Task.NrOfCycles);
end;

procedure TSystem.AllocateSelf(c:integer);

var s:integer;

begin
  for s:=0 to Agent[0].NrOfSkills-1 do
  begin
    if Agent[0].Skill[s].Active=true then
```

```
    repeat
      MakeEnvironment(s);
      if ParForm.ProcesBox.Checked= True then ShowMessage('Environment');
      Influence(s);
      if SysIntTime>5000 then SameAs(s);   //500 voor results simpatpaper
      ParForm.IntGraph[c+1]:=SysIntTime; //grafiek!!
    until CountIdoAgents(s)=1;
  end;
  FinalState(c);
end;

procedure TSystem.MakeEnvironment(s:integer);

begin
  if CountIDoAgents(s)> 1 then MakeIEnv(s);
  if CountIDoAgents(s)=0 then  MakeYouEnv(s);
  if ParForm.ProcesBox.Checked=True then ParForm.ShowStatesAgents;
end;

procedure TSystem.MakeIEnv(s:integer);

var a1,a2,t:integer;
    I,Y:real;
    DiffII, DiffYouYou:real;

begin
  for a1:=0 to NrOfAgents-1 do
  begin
    if Agent[a1].Skill[s].Choice=True  then  //agent
    begin
      I :=0; Y :=0; t:=0;
      DiffII:=0; DiffYouYou:=0;
      for a2:=0 to NrOfAgents-1 do
      begin
        if (a1<>a2) AND (Agent[a2].Skill[s].Choice=True) then  //omgeving
        begin
          t:=t+1;
          I:= I + (Agent[a2].Skill[s].I);
          Y:= Y + (Agent[a2].Skill[s].You);
          DiffII    :=DiffII+     Abs(Agent[a1].Skill[s].I  -Agent[a2].Skill[s].I);
          DiffYouYou:=DiffYouYou+ Abs(Agent[a1].Skill[s].You-Agent[a2].Skill[s].You);
        end;
      end;
      if t=0 then t:=1;
      I:=I/t; Y:=Y/t;   //gewogen gemiddelde van alle I's en Y's
      Agent[a1].EnvSkill[s].DiffII     :=DiffII/t;
      Agent[a1].EnvSkill[s].DiffYouYou :=DiffYouYou/t;
      Agent[a1].EnvSkill[s].I  := I;
      Agent[a1].EnvSkill[s].You:= Y;
      //Agent[a1].EnvSkill[s].Power:=t/10;
      Agent[a1].EnvSkill[s].Power:=1/10;
      Agent[a1].EnvSkill[s].Choice:=True;
    end;
  end;
end;

procedure TSystem.MakeYouEnv(s:integer);

var a1,a2,t:integer;
    I,Y:real;
```

```
begin
  for a1:=0 to NrOfAgents-1 do  //agents
  begin
    I :=0; Y :=0; t:=0;
    for a2:=0 to NrOfAgents-1 do  //omgeving
    begin
      if a1<>a2 then
      begin
        t:=t+1;
        I:= I + (Agent[a2].Skill[s].I);
        Y:= Y + (Agent[a2].Skill[s].You);
      end;
      if t=0 then t:=1;
      I:=I/t; Y:=Y/t;  //gewogen gemiddelde van alle I's en Y's
      Agent[a1].EnvSkill[s].I  := I;
      Agent[a1].EnvSkill[s].You:= Y;
      //Agent[a1].EnvSkill[s].Power:=t/10;
      Agent[a1].EnvSkill[s].Power:=1/10;
      Agent[a1].EnvSkill[s].Choice:=False;
    end;
  end;
end;

procedure TSystem.Influence(s:integer);

var a:integer;

begin
  for a:=0 to NrOfAgents-1 do
  begin
    if (CountIDoAgents(s)<>1) then
    begin
      Agent[a].RememberOldValues(s); //voor SameAs
      if Agent[a].Skill[s].Choice=True then Agent[a].InfluenceAgent(s);
      if CountIDoAgents(s)=0 then Agent[a].InfluenceAgent(s);
      if Agent[a].IntTime>SysIntTime then SysIntTime:=Agent[a].IntTime;
    end;
  end;
end;

procedure TAgent.InfluenceAgent(s:integer);

begin
  if (Skill[s].Expertise>Skill[s].ExpThresh) then
  begin
    if EnvSkill[s].Choice = TRUE then WeDo(s,  EnvSkill[s].DiffII);
    if EnvSkill[s].Choice = FALSE then YouDo(s,EnvSkill[s].DiffYouYou);
    if Skill[s].IntTime>IntTime then IntTime:=Round(Skill[s].IntTime);
  end;
end;

procedure TAgent.WeDo(s:integer; DiffII:real);

begin
  Skill[s].I:=   EnvSkill[s].Inhibit (EnvSkill[s].I, Skill[s].I, DiffII,  Skill[s].Inhibition,EnvSkill[s].Power);
  Skill[s].You:= EnvSkill[s].Excitate(EnvSkill[s].I, Skill[s].You, DiffII,
  Skill[s].Excitation,EnvSkill[s].Power);
```

```
  Skill[s].Choice:= Skill[s].I>Skill[s].You;
end;

procedure TAgent.YouDo(s:integer;DiffYouYou:real);

begin
  Skill[s].I:=   EnvSkill[s].Excitate(EnvSkill[s].You,Skill[s].I, DiffYouYou,
  Skill[s].Excitation,EnvSkill[s].Power);
  Skill[s].You:= EnvSkill[s].Inhibit( EnvSkill[s].You,Skill[s].You,
  DiffYouYou,Skill[s].Inhibition,EnvSkill[s].Power);
  Skill[s].Choice:= Skill[s].I>Skill[s].You;
end;

function TEnvSkill.Inhibit(x,y,I1,I2,I3:real):real;

begin
  Result:= y-(x*y*I1*I2*I3);
end;

function TEnvSkill.Excitate(x,y,I1,I2,I3:real):real;

begin
  Result:= y + x*(1-y)*I1*I2*I3;
end;
```